4. Consider the problem of keeping track of the available seats in a theater. Theater seats can be represented with a two-dimensional array of integers, where a value of 0 shows a seat is available, while a value of 1 indicates that the seat is occupied. For example, the array below shows the current seat availability for a show in a small theater.

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	1	1	0	1
[1]	0	1	0	1	0	1
[2]	1	0	0	0	0	0

The seat at slot `[1][3]` is taken, but seat `[0][4]` is still available.

A show can be represented by the `Show` class shown below.

```
public class Show
{
    /** The seats for this show */
    private int[][] seats;

    private final int SEATS_PER_ROW = < some integer value>;
    private final int NUM_ROWS = < some integer value>;

    /** Reserve two adjacent seats and return true if this was
     *  successfully done, false otherwise, as described in part (a).
     */
    public boolean twoTogether()
    { /* to be implemented in part (a) */ }

    /** Return the lowest seat number in the specified row for a
     *  block of seatsNeeded empty adjacent seats, as described in part (b).
     */
    public int findAdjacent(int row, int seatsNeeded)
    { /* to be implemented in part (b) */ }

    //There may be instance variables, constructors, and methods
    //that are not shown.
}
```

(a) Write the `Show` method `twoTogether`, which reserves two adjacent seats and returns `true` if this was successfully done. If it is not possible to find two adjacent seats that are unoccupied, the method should leave the show unchanged and return `false`. For example, suppose this is the state of a show.

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	1	1	0	1
[1]	0	1	0	1	0	1
[2]	1	0	0	0	1	1

A call to `twoTogether` should return `true`, and the final state of the show could be any one of the following three configurations.

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	**1**	**1**	1	1	0	1
[1]	0	1	0	1	0	1
[2]	1	0	0	0	1	1

OR

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	1	1	0	1
[1]	0	1	0	1	0	1
[2]	1	**1**	**1**	0	1	1

OR

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	0	1	1	0	1
[1]	0	1	0	1	0	1
[2]	1	0	**1**	**1**	1	1

For the following state of a show, a call to `twoTogether` should return `false` and leave the two-dimensional array as shown.

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	1	0	1	1	0
[1]	1	1	0	1	0	1
[2]	0	1	1	1	1	1

Class information for this question

```
public class Show

private int[][] seats
private final int SEATS_PER_ROW
private final int NUM_ROWS
public boolean twoTogether()
public int findAdjacent(int row, int seatsNeeded)
```

Complete method `twoTogether`.

```
/** Reserve two adjacent seats and return true if this was
 *  successfully done, false otherwise, as described in part (a).
 */
public boolean twoTogether()
```

(b) Write the `Show` method `findAdjacent`, which finds the lowest seat number in the specified row for a specified number of empty adjacent seats. If no such block of empty seats exists, the `findAdjacent` method should return `-1`. No changes should be made to the state of the show, irrespective of the value returned.

For example, suppose the diagram of seats is as shown.

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	0	1	1	0	0	0
[1]	0	0	0	0	1	1
[2]	1	0	0	1	0	0

The following table shows some examples of calling `findAdjacent` for `show`.

Method call	Return value
`show.findAdjacent(0,3)`	3
`show.findAdjacent(1,3)`	0 or 1
`show.findAdjacent(2,2)`	1 or 4
`show.findAdjacent(1,5)`	-1

Complete method `findAdjacent`.

```
/** Return the lowest seat number in the specified row for a
 *  block of seatsNeeded empty adjacent seats, as described in part (b).
 */
public int findAdjacent(int row, int seatsNeeded)
```

STOP

END OF EXAM